Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 20:20

Outline

- The Weak-Duality between Matching and Cover
- The Hungarian Algorithm for Weighted Bipartite Matching
  - General Properties
  - Simple $O(n^4)$-time implementation
  - Sketch of $O(n^3)$-time implementation
- Concluding Notes
  - Maximum Weight Matching in General Graphs

The Weak Duality between Maximum Matching & Minimum Cover

The ***weight of minimum vertex cover*** is always <u>*at least*</u> the ***weight of maximum matching***.

The Maximum-Weight Matching Problem

- Input :
  - A graph $G = (V, E)$ with edge weight $w_{u,v}$ for all $(u, v) \in E$.
- Output :
  - A matching $M \subseteq E$ that has the maximum weight among all possible matchings in G.
    - That is, $\sum_{e \in M} w_e \geq \sum_{e \in M'} w_e$ holds for all matching M' in G.

The Minimum-Weight Vertex Cover Problem

- Input :
  - A graph $G = (V, E)$ with edge weight $w_{u,v}$ for all $(u, v) \in E$.

- **Definition**. ((Weighted) Vertex Cover)
  - A label (function) $y : V \longrightarrow \mathbb{R}$ is a vertex cover for G, if

    $y_u + y_v \geq w_{u,v}$ holds for all $(u, v) \in E$.

  - $w(y) \coloneqq \sum_{v \in V} y_v$ is defined to be the weight of y.

The Minimum-Weight Vertex Cover Problem

- Input :
  - A graph $G = (V, E)$ with edge weight $w_{u,v}$ for all $(u, v) \in E$.
- Output :
  - A vertex cover y for G that has the minimum weight among all possible vertex covers for G.
    - That is, $\sum_{v \in V} y_v \leq \sum_{v \in V} y'_v$ holds all vertex cover y' for G.

Lemma 1. (Weak-Duality between Matching and Vertex Cover)

Let $G = (V, E)$ be a graph with edge weight w_e for all $e \in E$,
M be a matching, and y be a vertex cover for G.

Then, $w(y) \geq w(M)$, i.e., $\sum_{v \in V} y_v \geq \sum_{e \in M} w_e$.

- The proof for Lemma 1 is straightforward.
  - Since the endpoints of edges in M are distinct, we obtain

$$\sum_{v \in V} y_v \geq \sum_{(u,v) \in M} (y_u + y_v) \geq \sum_{e \in M} w_e .$$

Remarks.

- Lemma 1 implies that,
  - If $w(y) = w(M)$ holds for some M and y, then they are both optimal.
  - In this case,
    we say that M and y *witnesses* the optimality of each other.
- The duality between matching and cover can appear in different forms for different problem models.
  - In this lecture, we examine the case on edge-weighted graphs.

The Weighted Matching Problem

in Bipartite Graphs

The Maximum Weight Bipartite Matching Problem

- Input :
  - A ***bipartite*** graph $G = (V, E)$ with ***partite sets $\boldsymbol{A}$ and $\boldsymbol{B}$*** and edge weight $w_{i,j} \in \mathbb{R}$ for $i \in A, j \in B$.

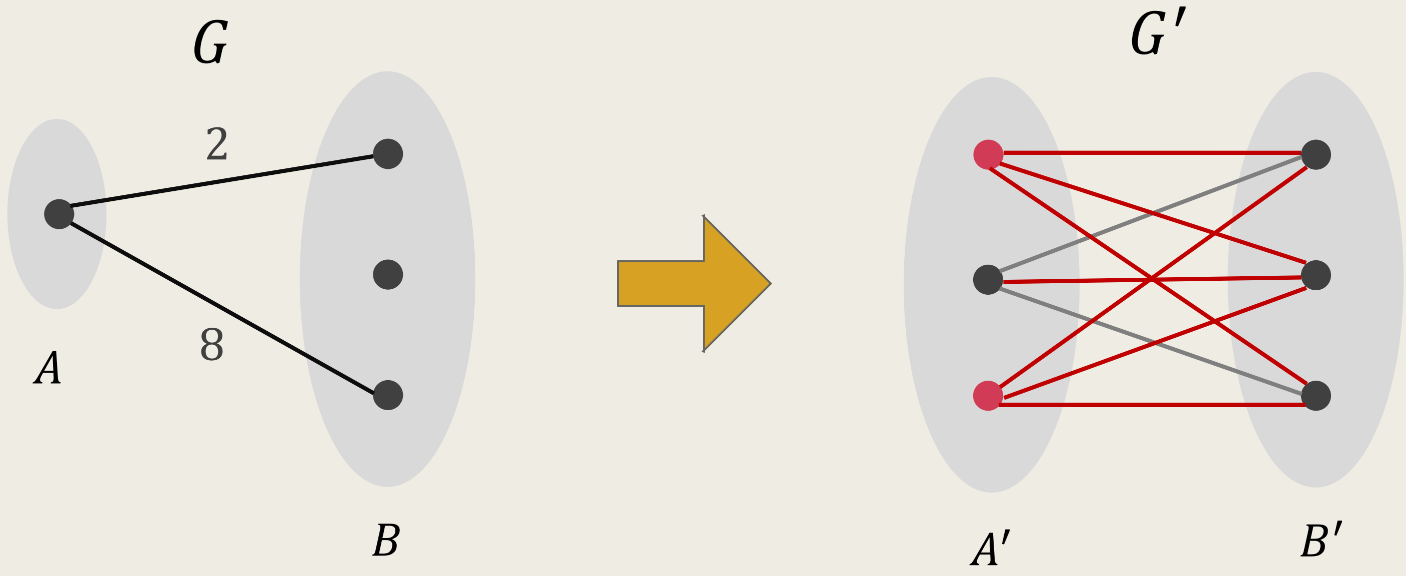
- Output :
  - A matching $M \subseteq E$ that has the maximum weight among all possible matchings in G.

Assumptions

- Without loss of generality, we may assume that…
  - $|A| = |B|$, and G is a complete bipartite graph.
    - If not, we add redundant vertices and edges with *sufficiently small weight* to make it so.
    - For example, the weight $\eta \coloneqq \min_{e \in G} w_e - 1$ will do.

Assumptions

Add redundant vertices and edges,
so that $|A'| = |B'|$, and G' is complete bipartite.



New edges have weight $\eta := \min_{e \in G} w_e - 1.$

- Without loss of generality, we may assume that…
  - $|A| = |B|$, and G is a complete bipartite graph.
    - If not, we add redundant vertices and edges with *<u>sufficiently small weight</u>* to make it so.
    - For example, the weight $\eta \coloneqq \min_{e \in G} w_e - 1$ will do.
    - Since $\eta < \min_{e \in G} w_e$, it is never better to replace an existing edge with a redundant edge.

Hence, a maximum weight matching in G corresponds to a maximum weight matching in the new graph G', and vice versa.

Assumptions

- In conclusion, we may assume that
  - $|\boldsymbol{A}| = |\boldsymbol{B}|$,
  - G is ***complete bipartite***, and
  - The goal is to compute a ***maximum weight <u>perfect matching</u>,***

    i.e., a maximum-weight matching such that

    every vertex in the graph is matched.

Remark.

- The considered problem is also *equivalent to* the ***minimum weight perfect matching*** problem.
  - When a minimum weight perfect matching is sought, then we take $w'_{i,j} = -w_{i,j}$ and solve the maximum weight perfect matching problem.

A minimum weight perfect matching w.r.t. w is a maximum weight perfect matching w.r.t. w', and vice versa.

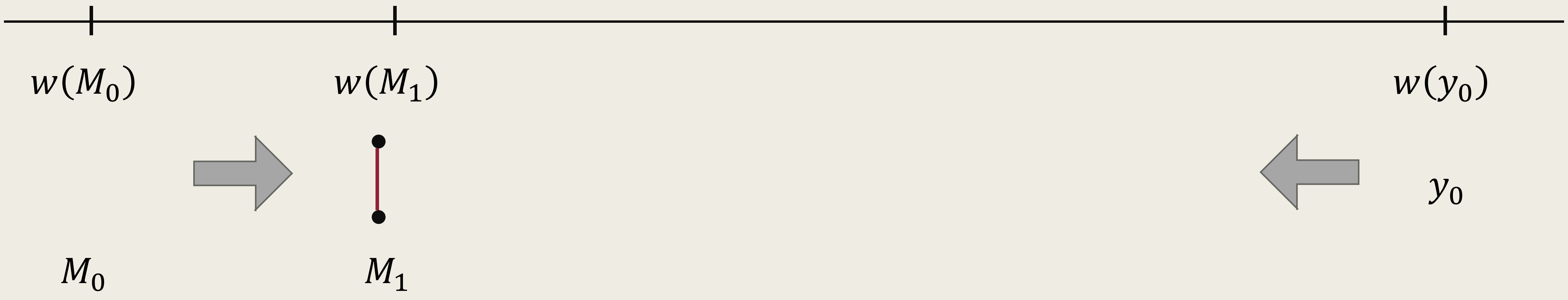
The Hungarian Algorithm

for Weighted Bipartite Matching

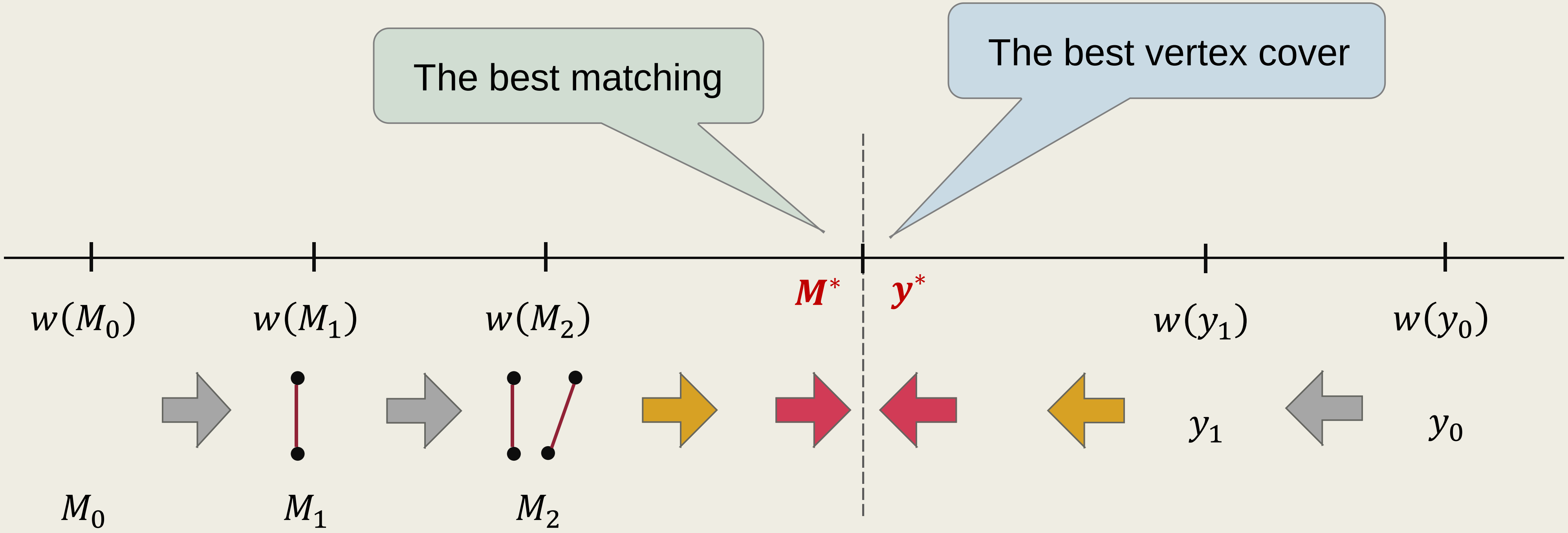
The Hungarian algorithm solves the problem via Primal-Duality of matching and cover.

The Hungarian Algorithm

- The algorithm starts with a trivial M and y.
  - In each iteration,
    the algorithm either improves M or y until their weights are equal.



- The algorithm starts with a trivial M and y.
  - In each iteration,
    the algorithm either improves M or y until their weights are equal.



The Hungarian Algorithm

- The algorithm starts with a trivial M and y.
  - In each iteration,
    the algorithm either improves M or y until their weights are equal.
    - We keep improving M,
      until it becomes unclear how M can be further improved.
    - Then we guarantee that,
      there must be a clear way to improve y.

The Hungarian Algorithm

- The Hungarian algorithm solves the weighted bipartite matching problem in $O(n^3)$ time.
  - We will first introduce the algorithm framework, which can be implemented in a simple way to run in $O(n^4)$ time.
  - Then we describe the $O(n^3)$ implementation of the algorithm.
    - It's more sophisticated, but
      can still be implemented in a nice and clean way.

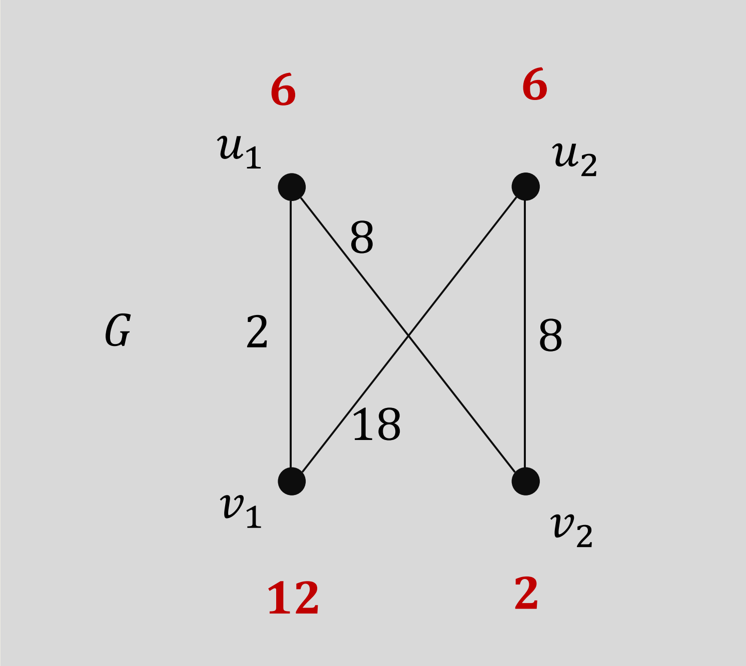
Key Notions and Properties

Equality Subgraph G_y

Defined according to the current y.

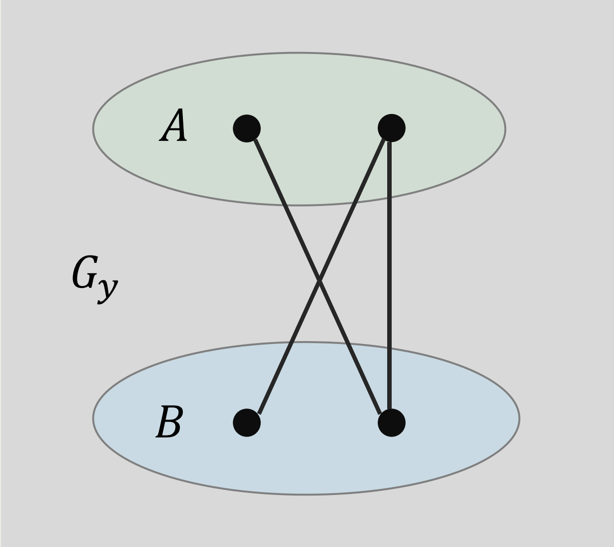
- Let y be a vertex cover for the input graph G.
  - Define the equality subgraph $G_y = (V, E_y)$ to be the graph with
    - Vertex set V
    - Edge set $E_y \coloneqq \{\ (u,v)\ :\ y_u + y_v = w_{u,v}\ \}$.

Intuitively, two vertices u and v are *connected in* G_y if and only if the ***weight*** y *uses to cover the edge* (u,v) *is* ***<u>the least possible</u>***.



$y_{u_1} = 6, \quad y_{u_2} = 6,$

$y_{v_1} = 12, \quad y_{v_2} = 2,$



- If there exists ***a perfect matching***, say, M, in G_y,

  then $w(M) = w(y)$ must hold, and both y and M are optimal for G.

The Goal – Looking for a Perfect Matching in G_y

- If we have a perfect matching for the equality subgraph G_y, then $w(M) = w(y)$ must hold,

  and both M and y are optimal by Lemma 1.

  - Hence, it suffices to come up with a y, such that G_y has a perfect matching.

  - How do we make this happen?

The Goal – Looking for a Perfect Matching in G_y

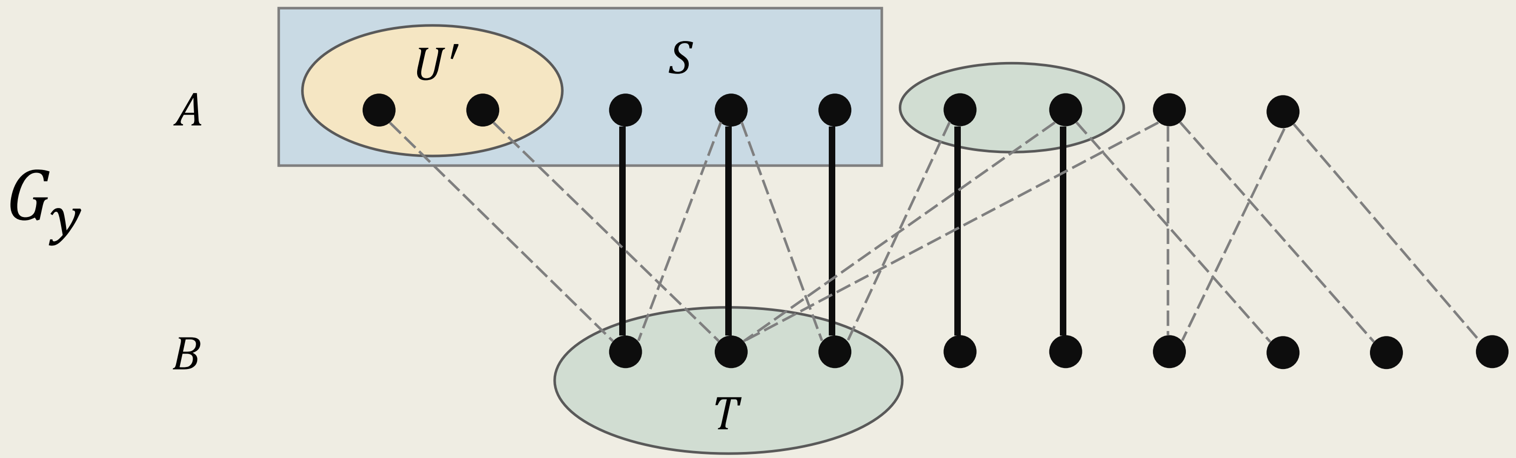
- Suppose that we have a vertex cover y and a matching M in the equality graph G_y.
  - Let $U \subseteq A$ be the set of unmatched vertices in A and $U' \neq \emptyset$ be an arbitrary nonempty subset of U.
  - Explore for M-augmenting paths for vertices in U' in G_y.
    - If found, then the size of M can be increased by 1.
    - If not…

- Consider a set U' of unmatched vertices.

  If there exists no M-augmenting path for U' in G_y, then…

  - Let S be the set of vertices in A that are reachable from U' via M-alternating paths.

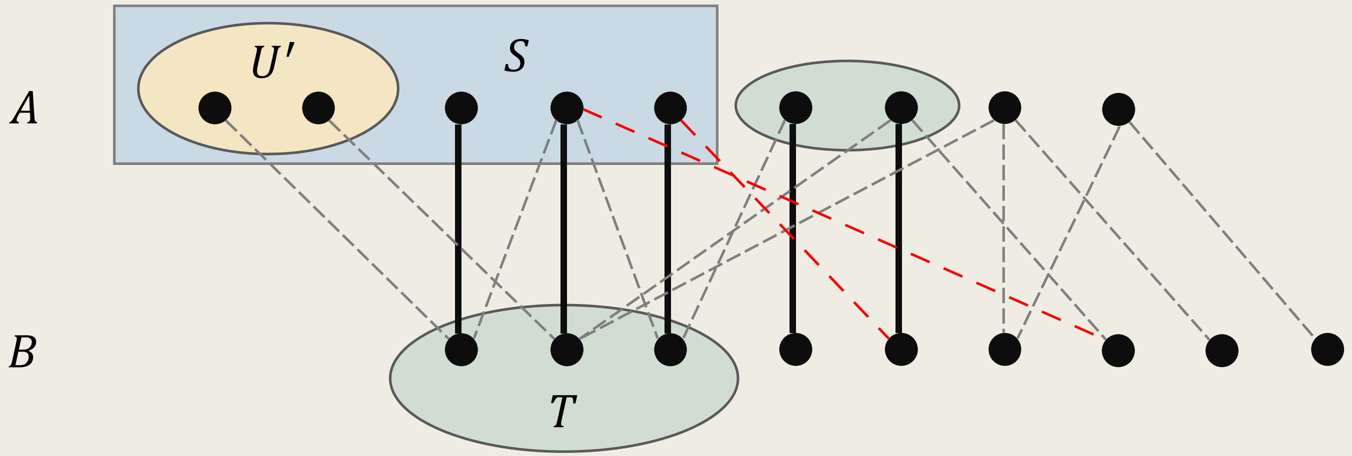
  - Let T be the set of vertices to which vertices in $S \backslash U'$ are matched by M.



Observations

- Since $|U'| > 0$, it follows that $|S| > |T|$.
- By the definition of S and T,
  there is no edge between S and $B \backslash \mathrm{T}$ in G_y.
  - In order to form an augmenting path for U',
    we need to create at least one edge between them.

By adjusting the vertex cover y properly.

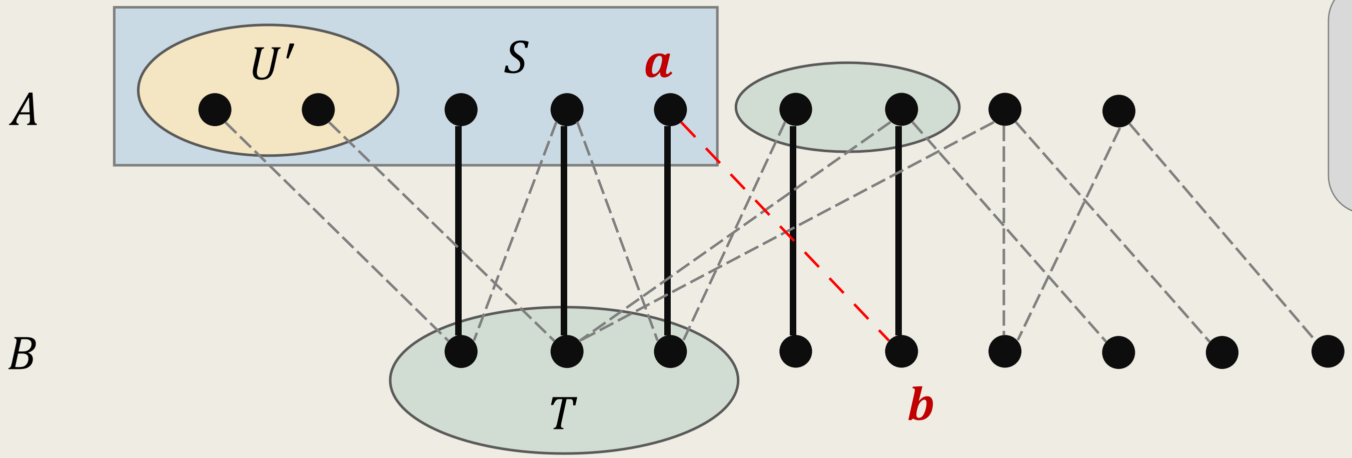


Adjusting the Cover y

while maintaining its feasibility.

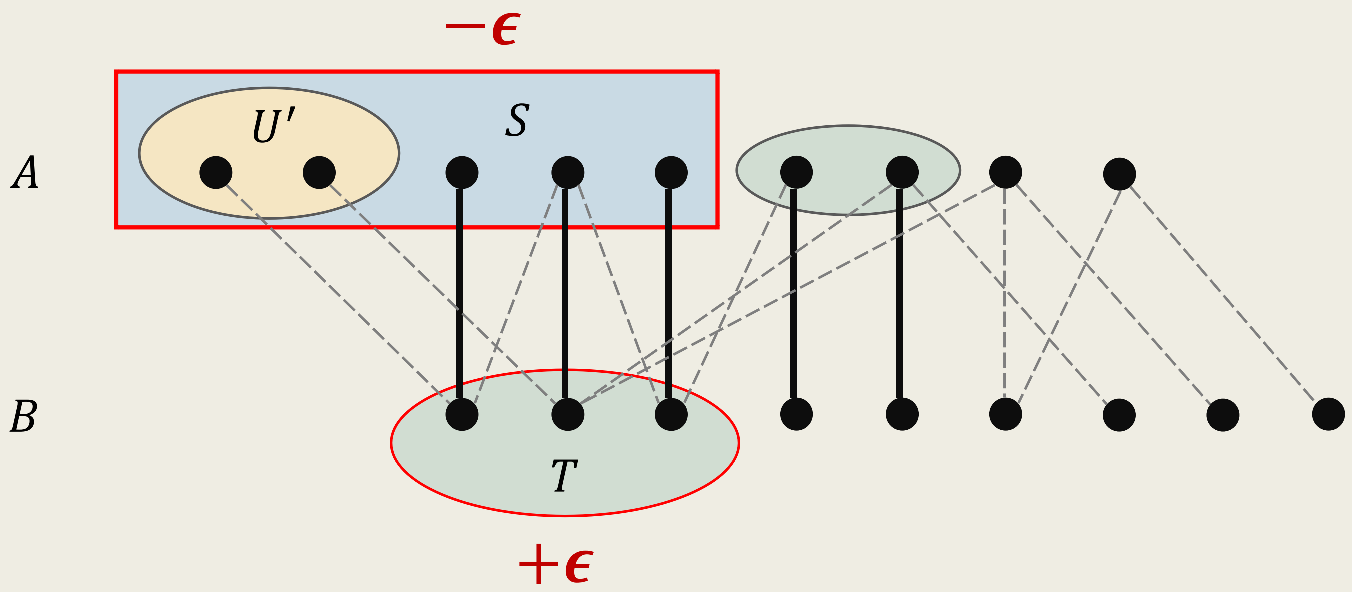
- For such an edge, say, (a, b), to appear in the equality graph G_y, where $a \in S,\ b \in B\backslash \mathrm{T}$,

  $y_a + y_b$ needs to be decreased by the amount $(y_a + y_b) - w_{a,b}$.



We call this the "***slack***" of edge (a, b).

This suggests the following procedure for adjusting y.

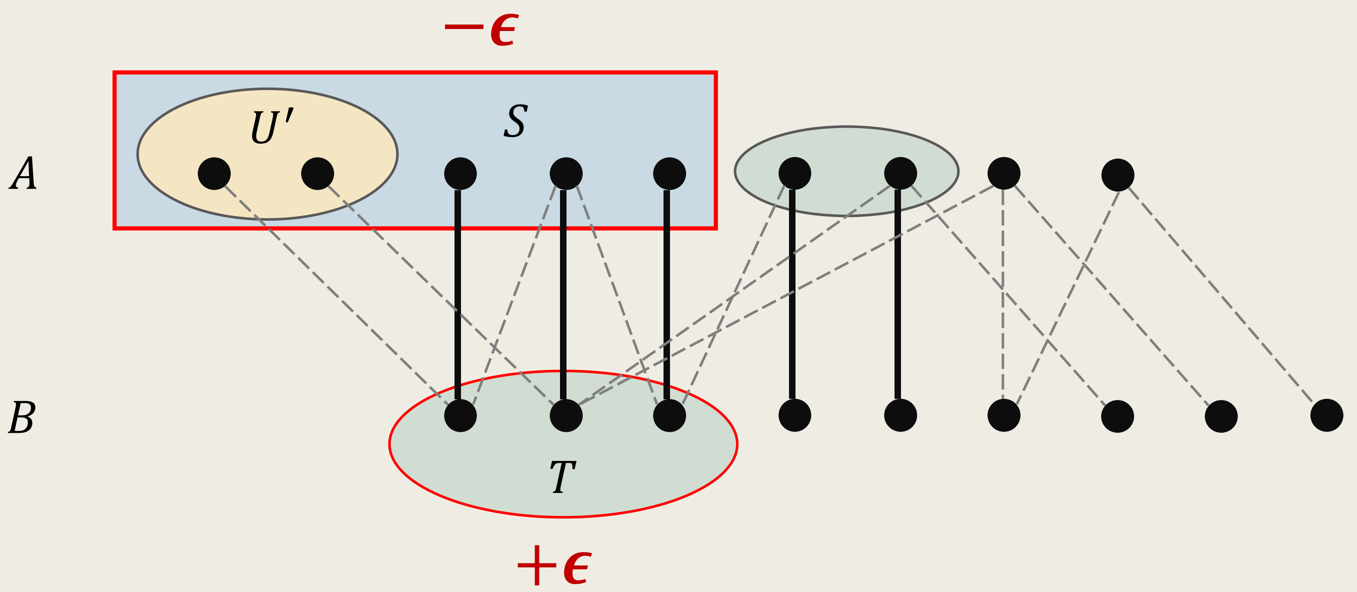


- Define $\epsilon = \min_{\substack{a \in S, \\ b \in B \setminus \mathrm{T}}} \left(y_a + y_b - w_{a,b} \right)$.

ϵ is the minimum “slack” of the edges between S and $B \setminus \mathrm{T}$.

- Observe that, if we
  - Decrease y_a by ϵ for all $a \in S$,
  - Increase y_b by ϵ for all $b \in T$,

The resulting y is still a valid vertex cover for G.



More vertices can be reached from U' via alternating paths.

- Then,
  - At least one edge between S and $B\backslash\mathrm{T}$ will appear in G_y.
  - Both the edges between S and T and the edges between $A\backslash\mathrm{S}$ and $B\backslash\mathrm{T}$ are unaffected.

All the matched edges remain in G_y.

- We lose the edges between $A\backslash S$ and T.

These edges play no role in M. So, we don't care.

The Adjusting Procedure on y w.r.t. U'

- Define $$\epsilon = \min_{\substack{a \in S, \\ b \in B \setminus T}} \left(y_a + y_b - w_{a,b} \right).$$
- Decrease y_a by ϵ for all $a \in S$ and increase y_b by ϵ for all $b \in T$. Then,
  - y remains a valid vertex cover for G.
  - The edges in M remain in G_y.
  - More vertices can be reached from U' via M-alternating paths.
- Since $|S| > |T|$, $w(y)$ is strictly decreased by $\epsilon \cdot |U'|$.

Looking for an Augmenting Path in G_y

- When y is adjusted,
  at least one edge between S and $B \backslash \mathrm{T}$ appears anew in G_y.

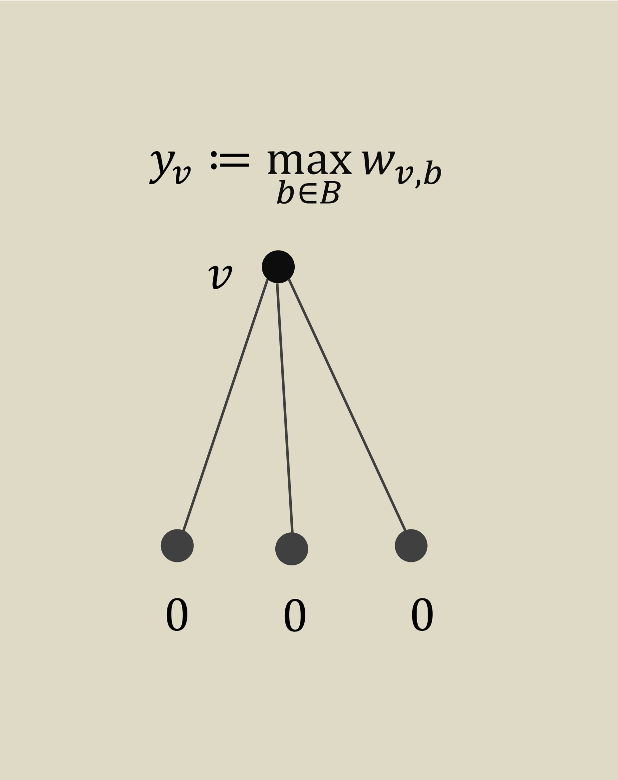
- Then, we **continue** to explore for M-augmenting paths for U'.
  - If found, the size of M can be increased by 1.
  - If not, we repeat the above procedure and adjust y
    until an M-augmenting path is found for some vertex in U'.

Description of the Algorithm

The Hungarian Algorithm

- The algorithm starts with $M = \{\emptyset\}$ and y defined as

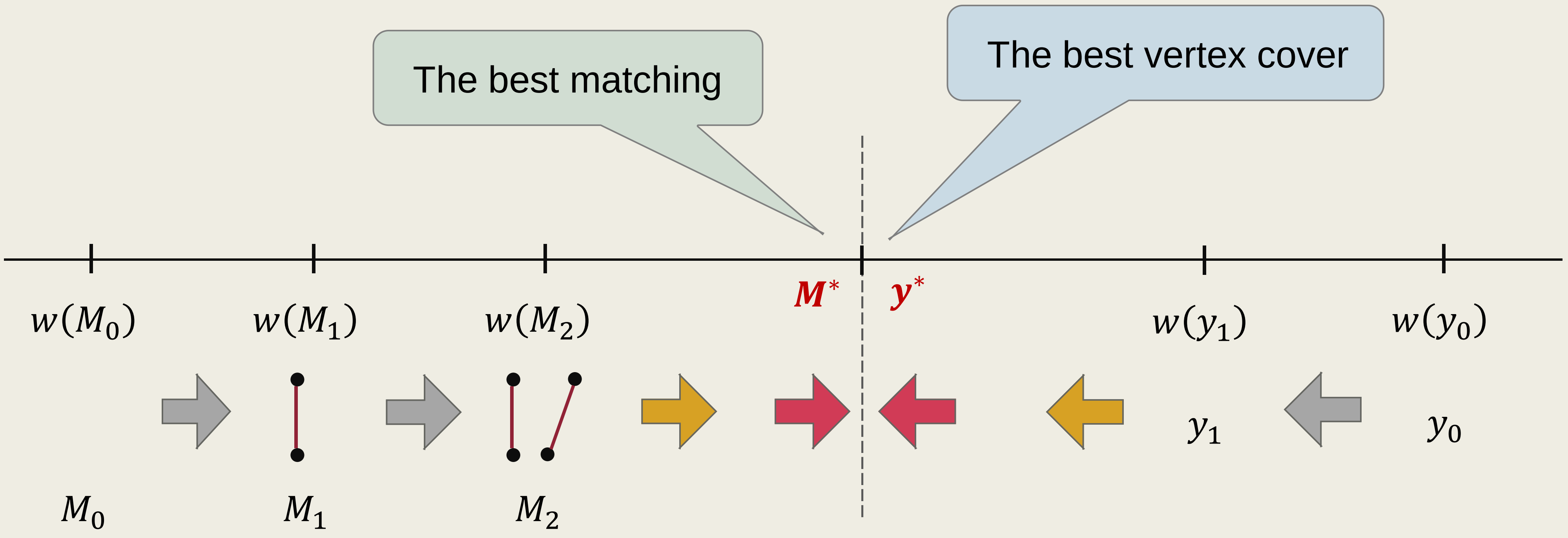
$$y_v := \begin{cases} \max_{b \in B} w_{v,b} \ , & \text{if } v \in A, \\ 0, & \text{if } v \in B. \end{cases}$$



It is easy to verify that the initial y is a feasible vertex cover for G.

- Repeat the following, until $|M| = n$.
  - Pick an unmatched vertex $v \in A$.
  - Repeat the following,
    until an M-augmenting path P for v in G_y is found.
    - $S \leftarrow$ vertices in A, reachable from v via M-alternating paths in G_y.
      $T \leftarrow$ vertices in B, to which vertices in $S \backslash \{v\}$ are matched by M.
    - Compute $\epsilon = \min_{a \in S,\, b \in B \backslash \mathrm{T}} \left(y_a + y_b - w_{a,b}\right)$.
      Decrease y_v by ϵ for all $v \in S$ and increase y_v by ϵ for all $v \in T$.
  - Use P to match v and increase $|M|$ by 1.
- Output M and y.

- The algorithm starts with a trivial M and y.
  - In each iteration,
    the algorithm either improves M or y until their weights are equal.



Correctness of the Algorithm

- By the previous observation, when an M-augmenting path is not found, the current y can be improved, and $|T|$ strictly increases.
  - Since $T \subseteq B$, an augmenting path can be found in $O(|B|) = O(n)$ number of updates on y.
  - Hence, the size of M can be increased until $|M| = n$.

    In this case, M is a perfect matching in G_y, and both M and y are optimal.

Time Complexity of the Algorithm

- It takes n iterations to compute a perfect matching.
  - For each of the iteration, y is updated $O(n)$ times.
  - In total, it takes $O(n^2)$ updates on M and y before the algorithm terminates.

- If we use a straightforward way for updating y in $O(n^2)$ time, then the algorithm takes $O(n^4)$ time.
  - Later we will see that, the Hungarian algorithm can be implemented to run in $O(n^3)$ time.

Simple $O(n^4)$ Time Implementation

Hungarian Algorithm in $O(n^4)$ Time.

- If we use the recursive procedure Aug-Path() from Slides #8, then the implementation is very simple, done as follows.

- For each unmatched vertex $u \in A$, do the following.
  1. Mark all vertices as unvisited.
  2. Repeat the following, until the procedure Aug-Path(u) on $G_y = \left(V, E_y\right)$ returns true.
     - Adjust y.
     - Remark all vertices as unvisited.

Hungarian Algorithm in $O(n^4)$ Time.

- Since the Procedure Aug-Path() takes $O(n^2)$ time, this implementation takes $O(n^4)$ time.
- Note that, we don't need to construct G_y.
  - It suffices to ***traverse only tight edges*** during DFS or BFS.
- Also note that, the set S and T needed to update y is already given by the information stored during the calls to Aug-Path() (i.e., DFS or BFS).

Just need to figure it out carefully.

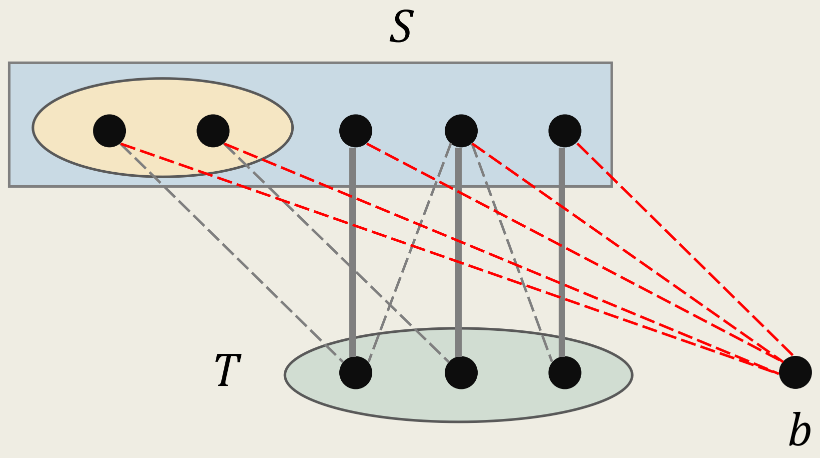
Sketch of

the $O(n^3)$ Time Implementation

Hungarian Algorithm in $O(n^3)$ Time.

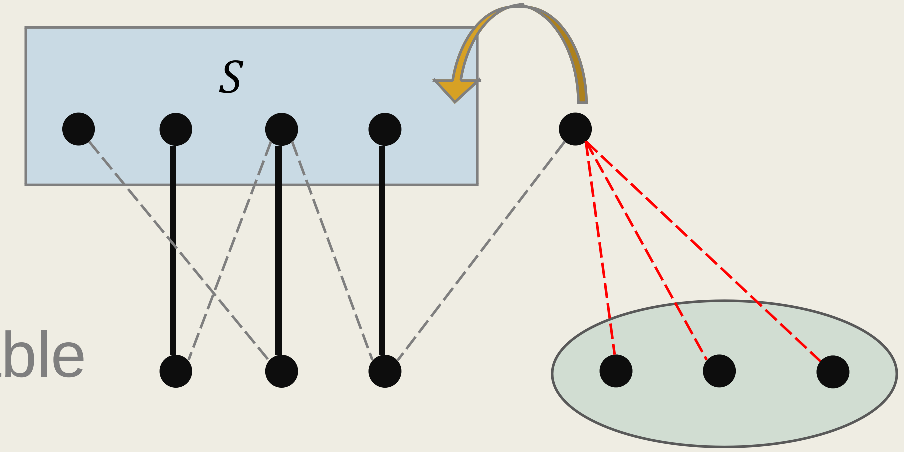
- Consider the algorithm framework in P.37.
  To make the algorithm run in $O(n^3)$ time,
  it is crucial that each iteration needs to be done in $O(n^2)$ time.
  - Since DFS or BFS already takes $O(n^2)$ time, it is important to ***continue*** from the ***currently unfinished exploration*** *each time when y is updated*, rather than restarting a new traversal.
  - Since y can be updated $O(n)$ times,
    the computation of ϵ needs to be done in $O(n)$ time.

Computing ϵ in $O(n)$ Time



- Recall that $\epsilon = \min_{a \in S,\, b \in B \setminus T} \left(y_a + y_b - w_{a,b}\right)$.
  - To speed up the computation, we define for each $b \in B \setminus T$ a slack variable
    $$\ell(b) \coloneqq \min_{a \in S} \left(y_a + y_b - w_{a,b}\right) .$$
  - Then ϵ can be computed in $O(n)$ time when needed, i.e.,
    $$\epsilon = \min_{b \in B \setminus T} \ell(b) .$$
  - The total time we spent for computing ϵ in each iteration is $O(n^2)$.

Computing ϵ in $O(n)$ Time



- Define for each $b \in B\backslash \mathrm{T}$ a slack variable

$$\ell(b) \coloneqq \min_{a \in S} \left(y_a + y_b - w_{a,b}\right) .$$

- The values $\ell(b)$ for all $b \in B\backslash \mathrm{T}$ need to be updated, *each time* when a new vertex is added to the set S during DFS or BFS.
  - This can be done in $O(n)$ time for each of such updates.
  - The total time it takes to update the values of $\ell(b)$ in each iteration is $O(n^2)$.

Concluding Notes

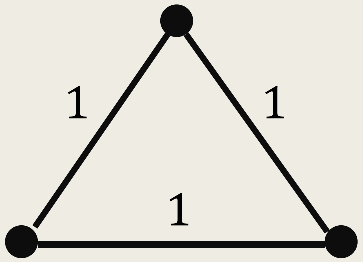
Maximum Weight Matching in Bipartite Graphs

- In this lecture, we introduced the Hungarian algorithm that solves the maximum weight matching and minimum weight vertex cover problems in bipartite graphs.

- The algorithm is also a constructive proof on the strong duality between matching and cover in bipartite graphs.
  - That is, $w(M^*) = w(y^*)$ must hold for any bipartite graph, whereas M^* and y^* are the optimal matching and vertex cover.

Maximum Weight Matching in General Graphs

- It is easy to see that, for general graphs,
  we do not have the strong duality between matching and vertex cover.
  - There are simple examples for which $w(M^*) < w(y^*)$.



- In fact, computing a minimum weight vertex cover in general graphs is an NP-hard problem.

Maximum Weight Matching in General Graphs

- However, strong duality still exists between matching and some combinatorial object, and it leads to a polynomial time algorithm.

- The maximum weight matching in general graphs can be computed by the Edmonds' Path-Tree-Flower algorithm in $O(n^2 m) = O(n^4)$ time.
  - The running time can be improved to $O(nm \log n) = O(n^3 \log n)$.
  - It is a generalization of the Blossom algorithm.